COMPUTER DIRECTORY SYSTEM HAVING AN APPLICATION INTEGRATION DRIVER INFRASTRUCTURE

Inventors: Nick N. Nikols
12067 Heron Ridge Circle
Draper, Utah 84020

Daniel Wallace Rapp
681 E. 200 N.
Alpine, Utah 84004

Assignee: Novell, Inc.
122 East 1700 South
Provo, Utah 84606

09470645.122299

HAYNES AND BOONE, L.L.P.
901 Main Street, Suite 3100
Dallas, Texas 75202-3789
(214) 651-5000
Haynes and Boone Docket No.26530.6
Novell Docket No. IDR:355

Attorney Docket No. IDR:355

EXPRESS MAIL NO.: EJ856342728US DATE OF DEPOSIT: 12/22/99

This paper and fee are being deposited with the U.S. Postal Service Express Mail Post Office to Addressee service under 37 CFR §1.10 on the date indicated above and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231

Karen A. Harris	[illegible]
Name of person mailing paper and fee	Signature of person mailing paper and fee

COMPUTER DIRECTORY SYSTEM HAVING AN APPLICATION INTEGRATION DRIVER INFRASTRUCTURE

Background Section

This invention relates generally to computer software, and more specifically to a system and method for allowing an application program to integrate and/or synchronize with a computer directory by providing a driver infrastructure for virtual replicas and the like.

Personal computers or workstations may be linked in a computer network to facilitate the sharing of data, applications, files, and other resources. One common type of computer network is a client/server network, where some computers act as servers and others as clients. In a client/server network, the sharing of resources is accomplished through the use of one or more servers. Each server includes a processing unit that is dedicated to managing centralized resources and to sharing these resources with other servers and/or various personal computers and workstations, which are known as the "clients" of the server.

Directories and directory services are often provided to enable a digital environment for a particular resource. One example of a directory service is Novell Directory Services ("NDS") for Novell Netware networks, as provided by Novell, Inc. of Provo, Utah. NDS provides a logical tree-structure view of all resources on the network so that clients can access them without knowing where they are physically located. For a given resource, an entry is made in a directory such as NDS. Specific entries of a directory are only available by directly accessing the directory.

Typically, a directory is established for one or more applications to access. A directory will have, for example, a native application program interface ("API") that provides a set of routines, protocols, and tools for using the directory. In this way, the directory's native API allows programmers to write applications that can utilize the information stored in a directory with appropriate access control. As a result, the applications must conform to the directory.

There has only been marginal success in the arena of application integration with directories. One of the primary problems is that many applications have already been written that utilize the same resources that are defined and managed from a directory. Also, many application programs were created prior to the advent of wide spread directory use. Of course any changes to application programs are inherently delicate, and are therefore undesirable.

Since these software applications cannot (or will not) access an entry as it resides in a particular directory, the applications will instead store information about the resource in their own directory-like data store. This creates a partial duplicate of the information which the application can access. Furthermore, the application can potentially augment the duplicated information. As a result, a given resource may be fractured among a number of applications or data repositories. As the number of applications that utilize a given resource increases, the likelihood of information about a given resource being unsynchronized, and therefore invalid, increases.

Replication is often used to synchronize distributed data within a given application in an automated and unattended manner. Replication enables many computers or computer applications to work with their own local, current copy, or replica, of a resource entry. For database applications where computers are widely distributed (e.g., geographically), replication provides an efficient way for distributed systems to access current information. To alleviate problems caused by partial data duplication, applications often try to maintain the validity of the replica by communicating with other applications that share the data stored in the replica. However, it is still important for the application to directly access the directory using the directory's native API or online directory service protocols like LDAP, or Lightweight Directory Access Protocol.

It is desired to provide centralized management of distributed resources to applications that utilize those resources, without requiring the applications to be rewritten to utilize the directory directly.

It is also desired to provide a directory interface that does not require updates or changes to an application program.

It is further desired to have a directory maintained as if multiple applications were working on a single, centralized directory.

Summary

In response to these and other problems, an improved system, method and software program is provided for facilitating the use of components running in a computer network. To this end, the improvement provides a driver infrastructure between components such as a distributed directory and a computer application. The driver infrastructure can transform specific directory events into a vendor-neutral data identification system and then use vendor-neutral transformation technologies to transform the neutral data identification into a specific application's data format, and vice-versa.

In one embodiment, the software program includes instructions for receiving an event from the distributed directory into a markup language generation system, such as an extensible markup language ("XML") generator. The XML generator converts the event into XML data and provides the XML data to a transformation system, such as an extensible transformation language ("XSLT") processor and/or a rule processor (generally "transformation processor"). The transformation processor

transforms the XML data to a predetermined format. The format can be dictated by a transformation profile, such as a stylesheet and/or one or more rules provided to the transformation processor, the transformation profile being responsive to requirements of the computer application. The transformed data is then provided to the application for the application to use in a conventional manner.

In some embodiments, the transformed data is transmitted to the application through an application shim. Some applications are able to accept input data streams while others are not. For applications in this latter group, the application shim may intercept the transformed data and provide it to the application by using a native application program interface for the application. In this way, if the application does not receive XML data, or some other interface necessity exists, the application shim can appropriately accommodate the application.

In some embodiments, the transformation processor can receive updates to the transformation profile. These updates, along with any changes to the application shim (if necessary), can accommodate any changes in either the distributed directory and/or the application. In this way, changes and/or updates to the distributed directory or the application can be accommodated in a separate, relatively simple manner.

The summary above discusses events going from the distributed directory to the application. However, the present invention also facilitates events going in the opposite direction. In one embodiment, a software program includes instructions for receiving an event from the application. A markup language generation system, such as an XML generator, converts the event into XML data and provides it to a transformation processor. The transformation processor transforms the XML data to a predetermined format required by the distributed directory. The transformed data is then provided to the distributed directory.

In some embodiments, the transformation processor going this opposite direction is the same as the previously mentioned transformation processor. To accommodate the different format requirements of the application and the distributed network, different transformation profiles can be used.

An advantage of the present invention is that it leverages existing text transformation technology to provide a general purpose data transformation.

Another advantage of the present invention is that a new or modified application needs simply to write a new application shim (if necessary) and provide the appropriate transformation profile. This provides easy integration between various applications.

These and other advantages can be readily seen by further examination of the attached figures and the following description.

Brief Description of the Figures

Fig. 1 illustrates a simplified computer system including two computers and a network, the system being used for implementing one embodiment of the present invention.

Fig. 2 is a diagram of an exemplary distributed directory system used in the computer system of Fig. 1.

Fig. 3 is a functional description of the computer system of Fig. 1 and the distributed directory system of Fig. 2. The functional description of Fig. 3 also illustrates a routine, such as can be implemented by a software program, for implementing one embodiment of the present invention.

Description of Embodiments

The present invention provides a unique system and method that invites integration between one or more applications and directories without requiring significant changes to the applications. It is understood that the following disclosure provides many different embodiments, or examples, for implementing different features. Techniques and requirements that are only specific to certain embodiments should not be imported into other embodiments. Also, specific examples of networks, components, and formats are described below to simplify the present disclosure. These are, of course, merely examples and are not intended to limit the invention from that described in the claims.

Referring now to Fig. 1, two similar computer systems, designated 10a and 10b, are illustrated as a representative example of an operating environment for the present invention. Each computer 10a, 10b includes a central processing unit ("cpu") 12a, 12b, a memory unit 14a, 14b, an input/output ("I/O") device 16a, 16b, and a network interface 18a, 18b, respectively. The components 12a, 14a, 16a, and 18a are interconnected by a bus system 20a, and the components 12b, 14b, 16b, and 18b are interconnected by a bus system 20b. It is understood that each of the listed components may actually represent several different components. For example, the cpu 12a may actually represent a multi-processor or a distributed processing system; the memory unit 14b may include different levels of cache memory, main memory, hard disks, and remote storage locations; and the I/O device 16a may include monitors, keyboards, and the like.

The computers 10a, 10b are also commonly connected to a network 30. The network 30 may be representative of several networks, such as a local area network, a company wide intranet, and/or the internet. Because the computers 10a, 10b are connected to the network 30, certain components may, at times, be shared between the computers. Therefore, a wide range of flexibility is anticipated in the configurations of the computers. Furthermore, it is understood that, in many implementations, the computers 10a, 10b may be configured differently from each other, may have different components, and/or one computer may act as a server to the other computer.

The present invention facilitates many different operational scenarios of the computers 10a, 10b and the network 30. Single server environments and multi-server environments, as well as distributed and non-distributed environments, may benefit from the present invention. A distributed, multi-server environment will be discussed below to provide just one example of how the present invention operates.

Referring now to Fig. 2, the computers 10a, 10b and the network 30 are used to provide a hierarchical distributed directory system 50. Software running on one or more of the computers 10a, 10b provide the directory service, while applications running on these computers (or other computers) utilize the directory service. The software may be stored on a recordable medium, such as one or both of the memory units 14a, 14b, the network 30, or other medium.

For the sake of example, the directory 50 is a NDS having a logical "tree-structure" view of all resources on the network. As a result, the computers 10a, 10b can access the resources without knowing where the resources are physically located (be it computer 10a, computer 10b, the network 30, or some other entity). For the sake of example, the directory 50 uses an online directory service protocol called LDAP, or Lightweight Directory Access Protocol. The directory includes one or more entries, each of which is a collection of attributes with a unique identifier.

In the present example, the directory 50 is broken into exclusive, non-overlapping "containers." A top level container A is connected to different lower containers 1, 2, 3, which are then connected to even lower containers a, b, c, etc. In furtherance of the present example, the top level container A may represent the overall directory structure for a large company; the containers 1, 2, 3 represent various cities that the company is located; and the lowest containers a, b, c represent different entities of the company, e.g., container a is for sales, container b is for marketing, and container c is for engineering. By combining the container names for more definite reference, sales 1a, 2a, 3a is in every city, marketing 1b, 3b is in two cities, and engineering 1c is only in one city.

One or more contiguous containers can be grouped into a single partition. A partition is a logical construct defined by software. Although a partition may be responsive to physical constraints (e.g., specific portions of a computer hard drive), it is not limited to a single physical location. In the present example, container A is in a partition 52; containers 1 and 1a are in a partition 54; container 1b is in a partition 56; containers 1c is in a partition 58; containers 2 and 2a are in a partition 60; and containers 3, 3a, and 3b are in a partition 62.

A computer, such as the computer 10a, can be configured as a server for storing one or more partitions and/or having a replica of one or more partitions. Replication synchronizes the distributed databases by routinely copying specific partition(s) to other servers in the network. Although replicas may be stored in one or more servers, any changes to data in a replica will replicate to the other servers storing the same data.

An application must be aware of the directory's structure, including any recent changes, upgrades, modifications, and so forth. Many applications are not created to access the directory structure (e.g., the application may not have been rewritten to utilize directory data access methods), and instead create a partial duplicate of this structure in their own directory-like data store. Because existing applications may not be rewritten (due to technical, financial, or other restraints) to directly access the directory, portions of the directory will be fractured among a number of applications or data repositories.

Referring now to Fig. 3, instead of forcing the applications to access the directory, the present invention brings the directory to the application. Fig. 3 provides a functional description of the computers 10a, 10b (Fig. 1) and the directory 50 (Fig. 2). Since the actual number and the physical arrangement of the computers, directories, and the associated applications is inconsequential, the functional description of Fig. 3 facilitates a broader understanding of the invention.

In continuance of the present example, a NDS server 100 interfaces with an application 102 residing in a system 104. Located between the server 100 and the system 104 is an application-integration driver infrastructure (designated generally "driver infrastructure") 106. It is understood that the NDS server 100, the driver infrastructure 106, and the system 104 may physically reside on one or more different computers.

In the present example, the driver infrastructure 106 is responsible for converting NDS replication events to XML (extensible markup language). XML is a common, flexible set of codes for indicating type, purpose, or structure of text in an application or directory, but is not specific to the application, directory, or even the specific vendor of the application or directory (i.e., it is vendor neutral). The driver infrastructure 106 also uses XSLT (Extensible Stylesheet Language Transformations) for defining actions that should be applied to the directory information XML to transform the directory information into an application native format. One example of directory information XML is Novell's DirXML, which is a technique that allows a user to create an application view of the information in the directory and then replicate that information via XML and an XSLT processor.

XSLT describes how the XML text is to be transformed into different XML text according to a formatting vocabulary. An extensible stylesheet language (XSL) defines a style, to be used by the XSLT, for specifying how to transform the XML text. XSL has two parts: a language for transforming trees, and a vocabulary of formatting objects. The application of XSL has heretofore been used only with publishing systems. The present invention applies XSL to directory integration for synchronizing multiple data repositories in a general fashion. The XML text can also be transformed by one or more rules and other transformation techniques. XML and XSL are just one example of generic data description, text conversion and transformation technology tools that can be used to implement the present invention.

The NDS server 100 includes an event handler 110, a driver interface 112, and a replica 114. The replica 114 represents at least a portion of the directory 50 (Fig. 2) that is stored in the NDS server 100. When a change is made to specific information (e.g., a phone number is changed somewhere in the directory 50), that change is replicated to the virtual replica 114. The event handler 110 is notified of the change and causes an event. In the present example, the event handler 110 receives filtered NDS events via an NDS Event system (not shown) and hands the events off to the driver interface 112. The event handler also checks for local transaction events and discards any events that should not be provided to the driver interface. The driver interface 112 picks up that event, validates the event (e.g., makes sure it is something to be handled), and then hands it off to the driver infrastructure 106.

The driver infrastructure 106 performs a routine to transfer the event to the application 102. At step 120, the event (formatted as NDS data) is provided to an XML generator 122, which converts the NDS data into XML. In continuance of the present example, an NDS event such as the change to the phone number is converted into NDS formatted XML.

At step 124, the XML data is provided to a transformation processor 126, which has a transformation profile that tells it how to transform the received information. The transformation profile may include a stylesheet and/or one or more rules. For example, the stylesheet may define a policy for the subset of NDS information that will be propagated to the application, and a data map for how a universal schema (e.g., NDS) is translated into an application specific schema. The transformation processor 126, which in this example includes an XSLT processor for utilizing the stylesheet, then transforms the format of the XML data to one appropriate for the application 102 (i.e., application data format). This application data format may be specific to the particular application 102, or may be another XML format.

For another example, one or more rules may be provided to the transformation processor 126 instead of, or along with the stylesheet. In this example, the transformation includes a rules processor to transform the format of the XML data to one appropriate for the application 102. In the present embodiment, the rules apply to XML, such as a schema mapping rule and or a matching rule. Formatting using rules such as these is often better suited than using XSL. It is understood that the transformation processor 126 may utilize an XSLT processor, a rules processor, both, or neither.

At step 128, execution returns to the XML generator 122 and at step 130, proceeds to an XSLT transmitter 132. Alternatively, execution may proceed directly from the XSLT processor 126 to the XSLT transmitter 132 at step 134. At step 136, the XSLT transmitter 132 passes the processed data to an application shim 138 located on the system 104. The processed data can be XSL, XML, LDIF. (LDAP Data Interchange Format), or any appropriate format.

The application shim 138 is a routine that bridges the gap between the driver infrastructure 106 and the application 102 by calling the application's APIs. In some embodiments where the application 102 already receives XML format, the application shim 138 may be unnecessary. Consider for example that the application 102 is a Netscape directory which has an API called LDAP modify, as provided by Netscape Comm. Corp. of Mountain View, CA. The LDAP modify API is capable of receiving and transmitting data in DIF. In this example, the XSLT processor 126 creates an DIF. data file, the XSLT transmitter 132 provides the file to the application shim 138, the application shim calls the LDAP modify API, and the LDAP modify API imports the DIF. data into the Netscape directory 102. In this way, the application shim 138 is totally dependent on the application 102.

The driver infrastructure 106 works equally well in the opposite direction -- from the application 102 to the NDS server 100. The application shim 138 must be aware of an event occurring in the application 102. This can happen by various methods, such as interrupts, polling, and so forth. If required, the application shim 138 can convert changes in the application's native directory into an application native XML format, and then submit that XML data stream to the driver infrastructure 106.

At step 140, the event (formatted as application data) is provided to an XML generator 142, which converts the application data into XML. At step 144, the XML data is provided to the transformation processor 126, which has a profile that tells it how to transform the received information. A separate transformation profile may be required to do transformations in this opposite direction, as compared to the profile for data going from the replica 114 to the application 102.

The XSLT processor 126 then changes the format of the XML data to one appropriate for the NDS server 100 (e.g., NDS data format). This NDS data format may be specific to the NDS server, or may be another XML format. At step 146, execution returns to the XML generator 142 and at step 148, proceeds to an XSLT receiver 150. Alternatively, execution may proceed directly from the XSLT processor 126 to the XSLT transmitter 150 at step 152. At step 154, the XSLT receiver 150 calls the APIs for the NDS server 100 and passes the processed data as NCP (Netware core protocol) requests to the NDS replica 114 located on the NDS server. The replica 114 then propagates the changes to any other replicas (if any exist) as replication events.

In the present example, the driver infrastructure 106 resides on the NDS server 100 (e.g., computer 10a of Fig. 1) and the application 102 resides on the system 104 (e.g., computer 10b of Fig. 1). The application shim 138 can reside on the NDS server 100, on the system 104, some other remote system, or split there between. If the application shim 138 is on a system other than the system 104, it can take advantage of remote APIs (like LDAP modify).

In summary, several different processing structures are disclosed, some or all of which may be utilized to implement the present invention. In the examples discussed above, the driver infrastructure is responsible for converting replication events to a set of codes for indicating type, purpose, or structure for application consumption. The driver infrastructure also receives a set of codes from the application and provides it to the server.

A server/driver infrastructure interface is responsible for the receipt of filtered events and converting them to the appropriate data format. It then hands the formatted data to a transformation processor. In the above example, this interface includes the XML generator 122, which converts NDS events into NDS formatted XML, and the XML receiver 150, which converts NDS formatted XML events into NCP requests to the NDS replicas 114.

A system/driver infrastructure interface is responsible for the transmission and receipt of transformed events to the application 102. If the application has an event interface and/or is able to accept the events directly, then this component is not required. Otherwise, as in the above examples, the application shim 138 will accept application native data stream from the driver infrastructure 106 and process it into application native APIs. It also will convert changes in the application native directory into an application native XML format, for submission to the driver infrastructure 106.

An advantage of the present invention is that it leverages a technology like XML to provide a general purpose transformation.

Another advantage of the present invention is that a new or modified application needs simply to write a new application shim (if necessary) and provide the appropriate transformation profile(s). This provides easy integration between various applications without incurring the prohibitive costs of rewriting the application.

Another advantage of the present invention is that by storing the policy and data mapping information within NDS by using a structure like the transformation processor 126, application integration is simplified.

It is further understood that other modifications, changes and substitutions are intended in the foregoing disclosure and in some instances some features of the disclosure will be employed without corresponding use of other features. Accordingly, it is appropriate that the appended claims be construed broadly and in a manner consistent with the scope of the disclosure.